

March 2025

8 Queens

An implementation on the RCA 1802 CPU

Non recursive/backtracking, with looping

8 Queens

An implementation on the RCA 1802 CPU

Non recursive/backtracking, with looping

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description/Changes** |
| 0.9 | March 2025 | Richard van Harderwijk | Good enough version |
|  |  |  |  |

Content

[Introduction 4](#_Toc192104836)

[Wait what? TI-59 design environment? 5](#_Toc192104837)

[Algorithm 6](#_Toc192104838)

[Nassi-Shneiderman 6](#_Toc192104839)

[Good Field? 6](#_Toc192104840)

[DSZ (Decrement and Skip on Zero) and Label naming 7](#_Toc192104841)

[Explanation from the TI-59 documentation 7](#_Toc192104842)

[Next additions 7](#_Toc192104843)

[Annex A – References and Tools used 8](#_Toc192104844)

[Annex B – The OS-ish - Utility UT4b 9](#_Toc192104845)

[Annex C – Example runs (input/output) 10](#_Toc192104846)

[Annex D – Code 11](#_Toc192104847)

# Introduction

This document describes the design and coding of a classic: the 8 Queens problem. This problem is normally solved with backtracking (a recursive, depth-first tree search). The code is designed for the RCA 1802 CPU with UT4 as OS-ish. The code runs on the RCA2020 [4] implementation of an 1802 SBC, but is easily portable to any 1802 system.

The solution is not based on backtracking but on loops. Background is the solution originally is designed for my TI-59 [5] and (by hand) cross compiled to the 1802.

# Wait what? TI-59 design environment?

Yes, if you think the 1802 is retro…

Afbeelding met rekenmachine, elektronica, tekst, Kantoorapparatuur

Door AI gegenereerde inhoud is mogelijk onjuist.

The TI-59 has only 6 levels of subroutines, 960 bytes total memory (code+data), 4 bit system and runs on 320 kHz. Due to that limited number of subroutine return depth, recursive was not possible and I had to come up with something else. It works, can take up to 40 minutes on the TI-59 (just a few seconds max on the 1802, mostly less, not noticeable).

And one day I took a notepad and pen just as I did for the TI-59 code, and cross-compiled the TI-59 to 1802 code by hand. Enjoy.

# Algorithm

## Nassi-Shneiderman

Afbeelding met tekst, schermopname, lijn, ontvangst

Door AI gegenereerde inhoud is mogelijk onjuist.

Old fashion code on old fashion machines deserve old fashion (but still handy) design tools.

My hand written flow charts translation from this is perhaps better not sharing.

## Good Field?

For a Queen to be on a good field, there cannot be another queen on the same row, column or the diagonals.

The rows and columns are numbered 1..8:

* If the row, column numbers of the potential place for the queen and already set queens are the same, the field is not valid.

The two diagonals are tested by evaluating:

* If the row+column number (adding) for both queens are the same, the field is not valid.
* If the row-column number (subtracting) for both queens are the same, the field is not valid.

## DSZ (Decrement and Skip on Zero) and Label naming

Looping on the TI-59 works with a variable, it is decreased by one every loop and jumped to a label. If the value of the counter is zero, then the jump is skip’d and the program is continued. So this is also used in the 1802 implementation.

The labels of the TI-59 are the buttons (mathematical operations). For sentimental reasons (and handy cross-compiling) I just kept the same label names.

## Explanation from the TI-59 documentation

* Backtracking is a depth-first search algorithm, very powerful because if a branch doesn’t come to a solution, you don't explore it further down (only a waste of time because you already know there is no solution).
* In the 8Q for example, you can put all queens on the same row. It is a solution, but after the second queen on the same row, you already know it is of no use trying to put the other 6 queens around the board.
* So what you do with backtracking, if you go down in the search tree and you know there can never be a solution: you go up one level in the three (backtrack) and go to the next branch. Etc.
* That can be implemented very elegantly with recursion (but with only 6 levels of subroutines…)

Emulate that with ‘for' loops. You start with input: a gueen position (row.column number 1-8; for example 6.4 = row 6, column 4)

* Then you have a loop for the other 7 q's
* You check if the row is free
* If the row is not free, you don’t have to try the columns, but jump to DSZ next row
* If the row is free, loop for 8 columns
* Check if column and both diagonals are free
* (Check diagonals: R+K is identical and R-K is identical)
* If valid, put the queen and goto next (optional: print queen number as indication how 'deep’ you are in the pseudo backtrack)
* If there is no valid solution on that free row, force the DSZ loop a queen back and put that queen on the next column and search again
* If necessary, if queen already on last column, go 2 queens back and put that queen to next column (‘automatically’ by DSZ)
* If ready, print the solution
* Even the 8th queen can be moved if it is on an invalid position
* I use indirect addressing as pointers to the current queen-register (R01-R08) (on 1802 stored on memory locations)

## Next additions

Afbeelding met ontwerp, keukenaccessoires, overdekt, kunst

Door AI gegenereerde inhoud is mogelijk onjuist.

The solution works, but why stop? I got my Arduino with this 8 x 8 LED matrix. That is a nice way to show the solution with 8 Queens. Can be driven via a parallel output port of the 1802, but more work and fun will be to use the RCA2020’s I2C interface, so there ya have it…

I’ll keep you posted, might take a while.

And when copying in some results in this documentation, see Annex C – Example runs (input/output), it came to my mind to put an easter egg in. Row 4, column 2, what to print??

# Annex A – References and Tools used

**References**

[1] MPM-201 User Manual for the CDP1802 COSMAC Microprocessor <http://www.bitsavers.org/pdf/rca/1802/MPM-201A_CDP1802_User_Manual_1976.pdf>

[2] MPM-203 Evaluation Kit Manual for the RCA CDP1802 COSMAC Microprocessor <http://www.bitsavers.org/components/rca/cosmac/MPM-203_CDP1802_Evaluation_Kit_Manual_Sep76.pdf>

[3] Documentation of this 8 Queens on 1802 implementation

<https://github.com/richardvanharderwijk/1802-8Queens>

[4] Documentation of the target 1802 system

<https://github.com/richardvanharderwijk/1802-SBC-the-RCA2020>

[5] Original implementation on a TI-59

<https://github.com/richardvanharderwijk/TI59-8Queens>

**Tooling used**

A big thank you! to the people building and maintaining these tools:

HexFiend (Hex editor for MacOS)

<https://hexfiend.com>

CoolTerm (Terminal emulator)

<http://freeware.the-meiers.org>

1802 online assembler-asm80

<https://www.asm80.com/onepage/asm1802.html>

Structorizer

<https://structorizer.fisch.lu>

# Annex B – The OS-ish - Utility UT4b

See MPM-203 [2]. It is a very simple utility to load, save and run code. It also has some routines to read and type in a ‘teletype’ way and those routines are used in my program. Read a byte; type a byte; type an ascii char; delay (needed after read).

UT4 is an example of a bit-banging serial communication interface (software UART). Funky 70’s code, worth exploring. All in just ½ kB.

The RCA2020 memory map (the rest is RAM)

8kB EEPROM, placed at 0xC000-0xDFFF

|  |  |  |  |
| --- | --- | --- | --- |
| **Mem map** | **ROM start** | **ROM end** |  |
| 0xC000 | 0x0000 | 00FF | On reset->run code. Room for (long branch to) other initialization code. End with branch to UT4b (256 Bytes) |
| C100 | 0100 | 02FF | Start of UT4b (½ kB) |
| C300 | 0300 | 03FF | Start of utility codes (SCRT ao) (256 Bytes) |
| C400 | 0400 | 1FFF | Free (7kB of ROM) (0xC400-0xDFFF) |

# Annex C – Example runs (input/output)

I have compiled two binaries:

1. 'Only’ printing the solution.
2. With printing the backtracking depth. Here you can see solutions take some going back to change the position of already placed queens.

......\*$P0

......

......8-Queens problem.

......Enter first Queen position (two numbers 1..8 for row and column): 84

......Calculating:

......84

......78

......65

......53

......41

......37

......22

......16

......

......Ready. Press return.

......

......\*$P0

......

......8-Queens problem.

......Enter first Queen position (two numbers 1..8 for row and column): 44

......Calculating:

......44

......86

......73

......67

......52

......38

......21

......15

......

......Ready. Press return.

......

......\*$P0

......

......8-Queens problem.

......Enter first Queen position (two numbers 1..8 for row and column): 48

......Calculating:

......48

......87

......74

......62

......55

......31

......23

......16

......

......Ready. Press return.

......

......\*

====== with print on =====

......\*$P0

......

......8-Queens problem.

......Enter first Queen position (two numbers 1..8 for row and column): 84

......Calculating:

......07

......06

......05

......04

......04

......03

......02

......01

......

......84

......78

......65

......53

......41

......37

......22

......16

......

......Ready. Press return.

......

......\*$P0

......

......8-Queens problem.

......Enter first Queen position (two numbers 1..8 for row and column): 44

......Calculating:

......07

......06

......05

......04

......04

......03

......02

......04

......03

......02

......05

......04

......04

......04

......03

......05

......04

......04

......06

......05

......04

......03

......02

......04

......03

......03

......05

......04

......04

......06

......05

......04

......03

......02

......04

......07

......06

......05

......04

......04

......03

......04

......03

......05

......04

......03

......02

......04

......03

......02

......03

......02

......05

......04

......03

......06

......05

......04

......03

......02

......01

......

......44

......86

......73

......67

......52

......38

......21

......15

......

......Ready. Press return.

......

......\*

# Annex D – Code

For the ‘non-depth printing/solution only’ binary, the printing code in Main-Function, at label ‘reciproke’ and type just before the return (D5) is commented out.

See github [3] for assembled code and binaries

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;

; Main: 8Queens solution with loops (no backtracking/recursive)

; For 1802 CPU (RCA 2020 SBC)

; Version: 0.9

; Date: March 3, 2025

; Author: Richard van Harderwijk

;

; Doc: https://github.com/richardvanharderwijk/1802-8Queens

; Note: this code is cross-compiled by hand from my TI-59 implementation,

; which has no recursive capability, so looping with DSZ (Decrement and Skip on Zero)

; and 'strange' label names (mathematical functions)

;

; No fancy error checking in this code

;

; Uses: SCRT, UT4 (read, type, delay)

;

; Register use:

; R(0) not used

; R(1) not used

; R(2) X / SP

; R(3) PC

; R(4) SCRT call

; R(5) SCRT return

; R(6) SCRT arguments passing

; R(7) temp

; R(8) R(8).0 return value from functions

; R(9) pointer to 8 Queens row.column values

; R(A) R(A).0 candidate row

; R(B) R(B).0 candidate column

; R(C) UT4 use

; R(D) UT4 use: read hex, assembled into by READAH

; R(E) UT4 use

; R(F) UT4 use: R(F).1 holds input/output ASCII char

;

; Queens stored in byte at pointer R(9): row=high nibble; column=low nibble

;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.org $0000

variables:

UT4\_TYPE .set 0xC2A4

UT4\_TYPE2 .set 0xC2AE

UT4\_READAH .set 0xC23B

UT4\_START .set 0xC100

;

; enter here at $0000 with X, P = 0, 0 via UT4 (int disabled)

;

init:

LDI 0x7F ; init stack/X in R2

PLO 2

LDI 0xFE

PHI 2

SEX 2

;

LDI 0x00 ; init PC in R3

PHI 3

LDI 0x0E

PLO 3

SEP 3

;

LDI 0xC3 ; init SCRT call in R4, return in R5

PHI 4

PHI 5

LDI 0x01

PLO 4

LDI 0x11

PLO 5

;

LDI 0x7F ; init pointer to 8Queens position values in R9

PHI 9

LDI 0x08

PLO 9

;

;

; print first message

;

print\_begin\_msg:

LDI 0x00 ; init pointer to text to print

PHI 7

LDI 0x76

PLO 7

;

print1:

LDA 7 ; get byte to print and put in R(F).1

PHI F

BZ cont1 ; exit if byte = 0x00 (EOF)

SEP 4

dw UT4\_TYPE ; SCRT to UT4: TYPE @ 0xC2A4

BR print1

cont1:

;

;

; input first queen/ 2 x READAH

;

SEP 4

dw UT4\_READAH ; SCRT to UT4: READAH @ 0xC23B

SEP 4

dw UT4\_READAH ; SCRT to UT4: READAH @ 0xC23B

;

GLO D ; store 8th queen position in @R(9)

STR 9

DEC 9 ; decrement pointer to 7 the queen location

;

SEP C ; call UT4.DELAY (after READ), returns w/ SEP 3

db 0x7F ; constant for DELAY

;

; print second message

;

LDI 0x00 ; init pointer to text to print

PHI 7 ; print 'calculating' message

LDI 0xCE

PLO 7

;

print2:

LDA 7 ; get byte to print and put in R(F).1

PHI F

BZ cont2 ; exit if byte = 0x00 (EOF)

SEP 4

dw UT4\_TYPE ; SCRT to UT4: TYPE @ 0xC2A4

BR print2

cont2:

;

;

;

; calculate 8Q solution

SEP 4 ; SCRT to main routine

dw start\_calculation

;

;

; 8Q solution ready

; print solution: 8 x print hex pair + 0x0D, 0x0A

;

LDI 0x08

PLO 9

PLO 7

print\_8q\_loop:

LDN 9 ; print hex pair value

PHI F

SEP 4

dw UT4\_TYPE2 ; SCRT to UT4: TYPE2 @ 0xC2AE

LDI 0x0D

PHI F

SEP 4

dw UT4\_TYPE ; SCRT to UT4: TYPE @ 0xC2A4

LDI 0x0A

PHI F

SEP 4

dw UT4\_TYPE ; SCRT to UT4: TYPE @ 0xC2A4

DEC 9

DEC 7 ; DSZ (Decrement and Skip on Zero)/loop

GLO 7

BNZ print\_8q\_loop

;

; print third message

;

LDI 0x00 ; init pointer to text to print

PHI 7

LDI 0xE0

PLO 7

;

print3:

LDA 7 ; get byte to print and put in R(F).1

PHI F

BZ cont3 ; exit if byte = 0x00 (EOF)

SEP 4

dw UT4\_TYPE ; SCRT to UT4: TYPE @ 0xC2A4

BR print3

cont3:

;

;

;

end\_8Q:

LBR UT4\_START ; Jump to UT4

;

;

begin\_txt:

db 0x0D, 0x0A, "8-Queens problem.", 0x0D, 0x0A, "Enter first Queen position (two numbers 1..8 for row and column): ", 0x00

calculating\_txt:

db 0x0D, 0x0A, "Calculating: ", 0x0D, 0x0A, 0x00

end\_txt:

db 0x0D, 0x0A, "Ready. Press return.", 0x0D, 0x0A, 0x00

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;

; Main-Function: 8Queens - solution with loops (no backtracking/recursive)

; For 1802 CPU (RCA 2020 SBC)

;

; Pre:

; Post on success:

; Post on error:

;

; End with SCRT return D5

;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.org $0100

start\_calculation:

;

sqr:

LDI 0x08 ; candidate row = 8

PLO A

;

sqr\_root:

SEP 4 ; SCRT to function check\_valid\_row

dw check\_valid\_row

GLO 8

BZ yx ; if not valid\_row, then branch to next row

LDI 0x08 ; if valid\_row, continue, candidate column = 8

PLO B

;

pr:

GLO A ; put row.column in candidate queen @R(9)

SHL

SHL

SHL

SHL

STXD

IRX

GLO B

ADD

STR 9

;

SEP 4 ; SCRT to function check\_valid\_field

dw check\_valid\_field

GLO 8

BNZ reciproke ; if valid\_field, then branch to next row

GLO B ; if not valid\_field, continue

SMI 0x02

BDF logx ; is BGE

INC 9 ; one queen back ('backtrack')

LDN 9

ANI 0x0F ; get column of backtracked queen

SMI 0x02

BDF absx ; (BGE) if last column of backtracked queen?

INC 9 ; then another queen backtrack

;

absx:

LDN 9 ; put row.column of backtracked queen

SHR ; in candidate row and candidate column

SHR

SHR

SHR

PLO A

LDN 9

ANI 0x0F

PLO B

;

logx:

DEC B ; DSZ (Decrement and Skip on Zero): next column

GLO B

BNZ pr

;

yx:

DEC A ; DSZ (Decrement and Skip on Zero): next row

GLO A

BNZ sqr\_root

;

;

reciproke:

; ; optional, print backtrack info/depth of queen

;

GLO 9 ; print #Queen ('backtrack' depth)

PHI F

SEP 4

dw UT4\_TYPE2 ; SCRT to UT4: TYPE2 @ 0xC2AE

LDI 0x0D

PHI F

SEP 4

dw UT4\_TYPE ; SCRT to UT4: TYPE @ 0xC2A4

LDI 0x0A

PHI F

SEP 4

dw UT4\_TYPE ; SCRT to UT4: TYPE @ 0xC2A4

;

;

;

lnx:

DEC 9 ; DSZ (Decrement and Skip on Zero) queen

GLO 9

BNZ sqr

;

;

;

LDI 0x0A

PHI F

SEP 4

dw UT4\_TYPE ; SCRT to UT4: TYPE @ 0xC2A4

;

; ; solution ready,

SEP 5 ; return with SCRT D5

;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;

; Sub-Function: 8Queens - check\_valid\_row

; For 1802 CPU (RCA 2020 SBC)

;

; Pre:

; Post on success:

; Post on error:

;

; End with SCRT return D5

;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

check\_valid\_row:

LDI 0x08 ; init temp pointer to Queens - R(7)

PLO 7

GHI 9

PHI 7

;

sinus:

GLO 7 ; compare candidate (subtract), if reached the candidate Q

STXD ; then no false return occurred in the DSZ loop

IRX ; means true return

GLO 9

SD

BZ return\_true\_row

;

LDN 7 ; test on row candidate queen

SHR ; subtract result zero means other queen on the row

SHR ; return with false

SHR

SHR

STXD

IRX

GLO A

SD

BZ return\_false\_row

;

DEC 7 ; DSZ test next queen row

GLO 7

BNZ sinus ; when all rows tested/end of DSZ, return with true

;

;

return\_true\_row:

LDI 0x01 ; R(8).0 = 1 (true)

PLO 8

SEP 5

;

return\_false\_row:

LDI 0x00 ; R(8).0 = 0 (false)

PLO 8

SEP 5

;

;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;

; Sub-Function: 8Queens - check\_valid\_field

; For 1802 CPU (RCA 2020 SBC)

;

; Pre:

; Post on success:

; Post on error:

;

; End with SCRT return D5

;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

check\_valid\_field:

LDI 0x08 ; init temp pointer to Queens - R(7)

PLO 7

GHI 9

PHI 7

;

sto:

GLO 7 ; test if queens above candidate tested

STXD ; if yes, return true

IRX

GLO 9

SD

BZ return\_true\_field

;

LDN 7 ; test if queen on equal column

ANI 0x0F ; if yes, return false

STXD

IRX

GLO B

SD

BZ return\_false\_field

;

LDN 7 ; test1 diagonals by

SHR ; adding row + column number

SHR ; if the same for candidate and queen already set

SHR ; return with false

SHR

STXD

IRX

LDN 7

ANI 0x0F

ADD

STXD

IRX

GLO A

SD

STXD

IRX

GLO B

SD

BZ return\_false\_field

;

LDN 7 ; test2 diagonals by

SHR ; subtracting row - column number

SHR ; if the same for candidate and queen already set

SHR ; return with false

SHR ; (row1-column1) - (row2-column2) = 0

STXD ; <=> row1-column1-row2+column2 = 0

IRX

LDN 7

ANI 0x0F

SD

STXD

IRX

GLO A

SD

STXD

IRX

GLO B

ADD

BZ return\_false\_field

;

DEC 7 ; DSZ test next

GLO 7 ; if all tested/end of DSZ, return with true

BNZ sto

;

;

return\_true\_field:

LDI 0x01 ; R(8).0 = 1 (true)

PLO 8

SEP 5

;

return\_false\_field:

LDI 0x00 ; R(8).0 = 0 (false)

PLO 8

SEP 5

;

;